B9

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) Internati nal Patent Classificati n 6 :
H03M 7/30, 7/42

A1

(11) International Publication Number: WO 95/19662

(43) International Publicati n Date: 20 July 1995 (20.07.95)

(21) International Application Number: PCT/US94/14823

(22) International Filing Date: 21 December 1994 (21.12.94)

(30) Pri rity Data:
08/181,187 13 January 1994 (13.01.94) US

(71) Applicant: TELCO SYSTEMS, INC. [US/US]; 63 Nahatan Street, Norwood, MA 02062 (US).

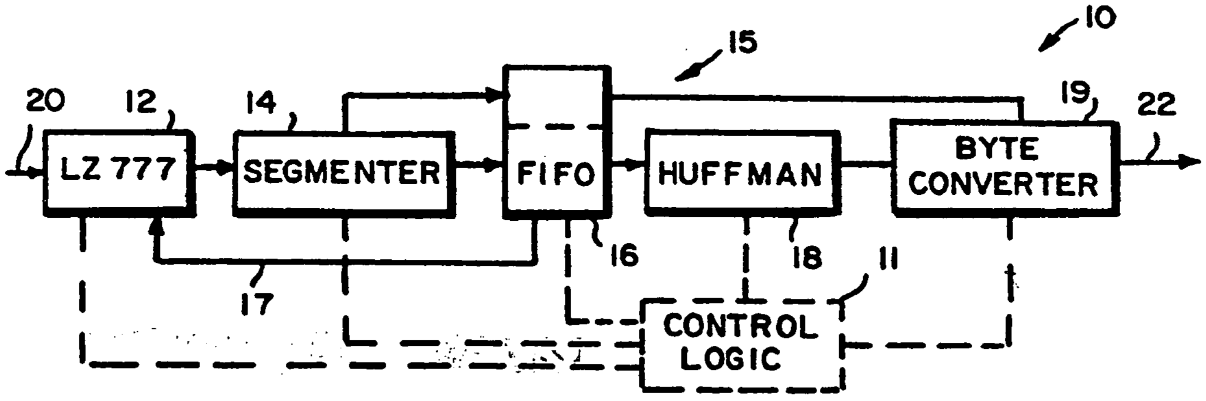
(72) Inventors: WEISS, Jeffrey, A.; 17 Maureen Drive, Smithfield, RI 02917 (US). SCHREMP, Douglas, H.; 1227 Gemini Drive, Annapolis, MD 21403 (US). SMITH, Aaron, T.; 50 East Manning, Providence, RI 02906 (US).

(74) Agent: ELBING, Kristofer, E.; Wolf, Greenfield & Sacks, P.C., 600 Atlantic Avenue, Boston, MA 02210 (US).

(81) Designated States: CA, JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: DATA COMPRESSION APPARATUS AND METHOD



(57) Abstract

Apparatus and method for compression of a digital sequence. The compressed sequence may be buffered and a compression procedure altered depending on buffer usage. This altering may include changing the depth of searching for matches performed in linked lists depending on the amount of the sequence being buffered. An adjacent element can be found in a linked list using a pointer, and a match can also be found in a history buffer using the same pointer. Frequency values may be updated for symbols based on a sampled subset of the symbols, and a token may be derived, such as by Huffman encoding, for each symbol from the updated frequency values. A subsequence may be encoded using an offset referenced to an earlier match. An insert pointer and a check pointer into a history buffer may be maintained, and a determination can be made as to whether accessing the history buffer is valid based on these check pointers. The maximum amount of the history buffer in which accessing is valid may be set. Decompression of sequences is also presented.

DATA COMPRESSION APPARATUS AND METHOD

Field of the Invention

The present invention relates to methods and apparatus for compression of digital sequences.

Background of the Invention

In digital systems, it is often advantageous to compress digitally stored sequences, such as data sequences. These compressed sequences can be stored using less storage space. In addition, they can be transmitted over communication lines in less time, using less bandwidth, or using fewer communications resources than their uncompressed counterparts. Several approaches for achieving compression of digital sequences have been proposed.

These approaches have been divided into "lossy" and "lossless" compression methods. Lossy compression does not include all of the original information in its compressed output. This type of compression can be used in video and audio applications where approximations are tolerated by the human eye or human ear. Lossless compression methods produce compressed output that can be decoded to restore the original sequence exactly. These methods are thus generally more appropriate for digital computer applications, such as program files, where loss of information usually cannot be tolerated.

One general approach to lossless data compression is the ZIV-LEMPEL-77 (LZ77) method, presented in "A Universal Algorithm for Sequential Data Compression," by Jacob Ziv and Abraham Lempel, IEEE Transactions on Information Theory, Vol. IT-23, No. 3, May 1977, and its variants. In this type of compression, sub-sequences of symbols in a sequence that have occurred at an earlier point in the sequence may be encoded as references to these earlier occurrences. For example, a word may be encoded as an offset, which indicates where an earlier occurrence of the word may be found, and a length, which indicates the number of characters in that earlier occurrence of the word.

Another approach to data compression is known as Huffman encoding. This type of encoding uses statistical information about the frequency of occurrence of individual symbols within a sequence to develop a code for this sequence. Symbols that occur frequently in the sequence are encoded with shorter codes, and symbols that occur rarely are encoded with longer codes.

This type of encoding method may either use predefined statistical information (e.g., about the English language) or it may adaptively develop its statistical information during operation. One known method for developing such statistical information is to monitor symbols passing through a compression system and develop a tree representing the statistical distribution of the symbols within the sequence. This method requires some computational overhead, but allows the encoding method to adapt to changes in the statistical content of the data. For further introductory material on the subject of data compression, see "Putting Data on a Diet", by Douglas H. Schremp and Jeffrey A. Weiss, IEEE Spectrum, Vol. 30, No. 8, pp. 36-39 (1993).

It is also known to perform more than one compression method in succession on the same data sequence. In particular, it is known to perform LZ77-type encoding followed by Huffman encoding.

Additional compression is possible, however, as are improvements in the hardware, software, storage, and/or computational time requirements for such compression. The present invention provides improvements in one or both of these areas.

Summary of the Invention

In one general aspect, the invention features compressing a digital sequence by encoding the sequence according to a compression procedure. The compressed sequence is buffered and the compression procedure is altered depending on buffer usage. This altering may include changing the depth of searching for matches performed in linked lists depending on the amount of the sequence being buffered. In another general aspect, the invention features compressing a digital sequence by maintaining a linked list of entries, which each include a pointer. An adjacent element can be found in this linked list using one of the pointers, and a match can also be found in a history buffer using the same pointer. In a further general aspect, the invention features compressing a digital sequence by sampling a subset of received symbols, and updating frequency values for the received symbols based on the sampled subset. A token is derived for each of the received symbols from the updated frequency values. Huffman encoding may be used to derive the tokens. In another general aspect, the invention features detecting, in a digital sequence, a first match between a first and a second subsequence and a second match between a third and fourth subsequence. The fourth subsequence is encoded using an offset referenced to the first match. In another general aspect, the invention features decompressing a digital sequence by receiving a first match token, and thereafter receiving a second match token including an offset. The position of a subsequence in a history buffer is determined based on the offset and the first match token, and the subsequence is substituted for the second match token in the sequence. In a further general aspect, the invention features compressing a digital sequence by maintaining an insert pointer and a check pointer into a history buffer. Characters are inserted at the insert pointer and the history buffer is accessed to search for matches in the history buffer. A determination is made as to whether the step of accessing the history buffer is valid based on the insert and check pointers. The maximum amount of the history buffer in which accessing is valid may be set.

Brief Description of the Drawings

FIG. 1 is a block diagram of a compression system according to the invention;

FIG. 2 is a diagram illustrating the storage of data in connection with the first (LZ77) encoding stage of the embodiment of FIG. 1;

FIG. 3 is a physical data storage diagram illustrating physical storage of the data for the first (LZ77) encoding stage of the decoder of FIG. 1;

FIG. 4 is a diagram of an exemplary random access memory structure for use in implementing the embodiment of FIG. 1;

FIG. 5 is a diagram illustrating tokens produced by the segmenter of FIG. 1;

FIG. 6 is a composite graph illustrating an exemplary probability distribution of the likelihood of matching a given string for an increasing offset within the history buffer, and corresponding segment sizes;

FIG. 7 is a diagram illustrating an exemplary output stream from the compression system of FIG. 1;

FIG. 8 is a block diagram of the last (Huffman) encoding stage of the embodiment of FIG. 1;

FIG. 9 is a block diagram of a decompression system according to the invention.

FIG. 10 is an illustrative flow chart presenting principles of operation of the first (LZ77) encoding stage of the embodiment of FIG. 1;

FIG. 11 is an illustrative flowchart presenting principles of operation of the adaptive linked list searching of the first (LZ77) encoding stage of the embodiment of FIG 1;

FIG. 12 is an illustrative flowchart presenting principles of operation of the last (Huffman) encoding stage of the embodiment of FIG. 1; and

FIG. 13 is an illustrative flowchart presenting principles of operation of the segmenter of the embodiment of FIG. 1.

Description of the Preferred Embodiment

Referring to FIG. 1, an exemplary encoding system 10 includes an LZ77-variant encoder 12, a segmenter 14, a coupling first-in-first-out (FIFO) buffer 16, a Huffman encoder 18, and a byte converter 19. One or more coded indications of fill level (i.e., half full, one-quarter full, etc.) of the FIFO buffer are fed back to the LZ77 encoder over one or more feedback lines 17. An offset data path 15 is provided between the segmenter and the byte converter via the FIFO buffer. The system 10 may be implemented with digital hardware and/or software.

In overall operation, the exemplary encoding system 10 receives an input digital sequence 20, compresses it, and provides it as a compressed output sequence to a communications channel 22. Generally, the encoding system uses the LZ77 encoder 12 to encode repeated sequences of characters each as an offset and a length. The segmenter 14 expresses these using a segmenting scheme, and provides the resulting segment-based offset and the resulting segment-based length to the coupling FIFO buffer. The coupling FIFO buffer receives both the segment-referenced lengths and any uncompressed literal data, and provides them to the Huffman encoder 18. The Huffman encoder encodes these segment-referenced lengths and literal data and supplies them to the byte converter in the form of tokens. The FIFO buffer supplies the buffered segment-based offset to the byte converter. The byte converter provides the tokens to the communications channel in a byte-delineated format. Control of the system is performed by control logic 11, which may be located within the individual modules, or may be centralized, as shown in FIG. 1. Although the system has been partitioned into functional subsystems, which will be described in more detail below, these do not necessarily correspond to strict structural limitations and, for example, all of the functions shown in FIG. 1 may be performed by one or more suitably programmed processors.

Referring to FIG. 2, the LZ77 encoder 12 includes a circular history buffer 24, hashing logic 27, and an open hashing data structure 26. The hashing data structure includes a hash table 28 and one or more linked lists 30, which lists are each linked to separate entries 36 in the hash table. Each linked list includes one or more list elements 31, and each element 31 in the linked lists includes a pointer 38 to its predecessor and a pointer 40 to is successor.

In overall operation of the LZ77 encoder, the history buffer 24 is supplied with eight-bit binary characters, and the hashing logic 27 and hashing data structure 26 are used to search for three-byte matches in the history buffer. While eight-bit characters are described in the present embodiment, the invention is, of course, applicable to other formats. There is also no requirement that matches be performed on three bytes to reap the benefits of the invention. Further parameter changes, minor changes in the sequencing of operations, and other variations are also possible without losing the benefits of the invention.

Referring to FIGS. 1, 2 and 10, operation of the LZ77 encoder will be discussed in more detail. Upon startup, the circular history buffer 24 is initialized by making a check pointer 32 equal to an insert pointer 34 (step 200 in FIG. 10). This can be done by setting both pointers equal to zero. At this point, the entire history buffer is deemed to include invalid data. Data to be compressed are then added to the history buffer at the location pointed to by the insert pointer (step 202). As these data are added, the insert pointer is separated from the check pointer (step 204).

Using these two pointers in this way allows startup to be performed rapidly, as the contents of the buffer need not be altered upon reset. This can be advantageous during resynchronization operations that may occur during operation of the encoder. These may be necessary, for example, if the encoder and decoder lose synchronization with each other due to errors on the communication channel. During each access to the history buffer in matching operations, however, the address of the location accessed should be checked against these pointers to determine if the access is valid (step 212, discussed below).

The check pointer 32 marks the end of the history buffer. As the history buffer is filled, the check pointer does not change. Once a predetermined number of characters M (e.g., 32K) have been inserted into the history buffer (see step 206), the check pointer begins to follow the insert pointer (step 208). This number of characters M may be a user-settable parameter, allowing the user to effectively set the size of the history buffer depending on the anticipated application. For example, where it is important that a high compression ratio be obtained, the user may set the history buffer to its maximum size. This feature may also allow systems using different amounts of physical memory to be operated together, with a system having more memory operating in a compatibility mode which sets a reduced history buffer size.

Each time a new character is inserted into the history buffer, this character is grouped with the two characters received immediately before it. The hashing logic 27 applies a hashing function (step 207) to these three most recently inserted characters each time a new character is inserted. An exemplary hashing function is:

KEY=((Byte1 XOR (Byte2 shifted left 3)) XOR Byte 3 shifted left 8) and KEYMASK

The resulting hash value is used to address the hash table 28 (step 210). If there is no entry 36 in the hash table that corresponds to this value (branch 211), this indicates that there is no match in the history buffer. A pointer to the location of the three-character sequence in the history buffer is then stored at that location (step

234), a character of literal data is provided to the segmenter (step 232), and the next character sequence is processed.

If the location addressed by the hash value contains an entry 36 (branch 213), this indicates that there could be a match in the history buffer 24. This entry is first checked to see if it points to a valid location by determining whether it is between the insert and check pointers (step 212). To check for a match, the three bytes pointed to in the history buffer by the pointer stored in the hash entry are then compared with the current three byte sequence which was provided to the hashing logic 27 (step 214). If these do not match (branch 215), the linked list that is linked to the addressed hash table entry will be searched for a match. This search will take place until A) a maximum length match (e.g., 64 bytes) is found (step 220), B) the linked list is exhausted (step 218, branch 222), C) an attempt is made to read outside of the history buffer (branch 224), or D) more than a certain number of linked list entries have been processed (step 225, branch 226), as discussed below.

If a detected match (branch 221) is shorter than 64 bytes long (branch 231), but longer than any previous match, it is saved in a current best match register as the current longest match (step 223). Then, once any searching in the hashing data structure is complete (branch 227), the current longest match is encoded as an offset and a length and provided to the segmenter 14 (step 230). The offsets can be expressed as the number of characters separating the beginning of the encoded match and the end of the earlier sequence that it matches. If no match has been detected (step 228), one character is provided to the segmenter as literal data (step 232).

The linked lists 30 have a maximum search depth, as indicated above by the fourth exit condition (D), which depth may be determined by a system parameter stored in a register or in memory. This parameter may be set to optimize average throughput for the encoder 12, or may be set adaptively. The use of this parameter can prevent large amounts of time being expended searching for old entries in the history buffer. In one embodiment, the default parameter is set to be 1024 entries.

When the system is used in its adaptive mode, referring to FIGS. 1 and 11, FIFO fill level information may be used at the encoder to determine the search depth parameter. For example, a "FIFO nearing empty" signal may be provided on the feedback lines 17 from the FIFO when it has fewer than a certain number of entries (step 240 of FIG. 11). This signal will then cause the encoder to reduce its search depth (step 242) and provide data to the FIFO 16 via the segmenter 14 more rapidly. In this mode, there is a chance that shorter than optimal matches will be found, but encoded data will be provided to the FIFO at a higher rate.

It is therefore possible to ensure that there will never be an interruption in transmission of the encoded characters, unless there is no more input data to process. This adaptive approach to data encoding is likely to provide a higher overall throughput than might otherwise be provided, although the level of compression may not be as high.
For this reason, this mode can be advantageous in communications applications.

Once the FIFO 16 has been replenished (step 244), because it has been filled faster than the communication channel could empty it, a "FIFO adequately filled" signal may be provided to the encoder 12 on the feedback lines 17. This signal will cause the encoder to increase its search depth again (step 246).

Referring to FIGS. 2 and 3, the LZ77 encoding will be discussed in further detail in connection with some example table entries. The encoder may be implemented using a series of four memory arrays having equal numbers of entries. The first array is the history buffer 24 (e.g., 32K x 8 bits), the second array is the hash table 28 (e.g., 32K x 16 bits), the third array is a predecessor array 42 (e.g., 32K x 16 bits), and the fourth array is a successor array 44 (e.g., 32K x 16 bits). The predecessor array and the successor array make up the storage area for the linked lists 30.

In a first example, a new three-byte combination in the history buffer 24 causes the hashing logic 27 to address a particular location 46 in the hash table 28. This location contains a pointer entry to a buffer entry 48 in the history buffer. As discussed above, the three bytes stored beginning at this buffer entry can now be tested to determine whether there is a match, by comparing them with the three-byte combination. If there is no match, and there is no element of a linked list associated with the table entry, a match is not found. A copy of the insert pointer is now added to the structure as the first element (not shown) of the linked list associated with that entry 46.

If the hashing logic addresses another particular hash table entry, for example table entry 50, which has a linked list associated with it, the buffer entry 52 pointed to by the contents of the table entry is similarly checked for a match. If no match is found, the entry 50 of the hash table is used to point to a predecessor entry 54 in the predecessor array 42, and its corresponding successor entry 56 in the successor array 44. Together, these two entries make up a first element of the linked list.

The predecessor entry 54 in the predecessor array points back to the hash table entry 50. The entry 56 in the successor array points to another location 58 in the history buffer which may be checked for another match. Advantageously, this successor entry also points to the next predecessor entry 60 and successor entry 62 (i.e., the second element in the linked list). Because entries in the linked list point to other entries in the linked list and to entries in the history buffer at the same time, each element of the linked list need only contain a predecessor and successor entry, and need not contain a separate pointer to the history buffer. This type of structure occupies less memory than might otherwise be required. The fact that the arrays are all the same size facilitates the implementation of this feature.

The predecessor entry 60 of the second element in the linked list points to the first element in the linked list, which includes its predecessor entry 54 and its successor entry 56. Note that it also points (not shown) to the history buffer entry 52 pointed to by the second hash table entry 50. The successor entry 62 of the second element in the linked list points to yet another buffer entry 64 in the history buffer 24, which may be checked for a match.

This successor entry 62 also points to a predecessor entry 66 and a successor entry 68 of a third linked list element. This third predecessor entry points back to the predecessor element 60 and the successor element 62 of the second linked list element. The successor entry of the third linked list element in this instance is an end marker, which indicates the end of the linked list.

Referring to FIGS. 3 and 4, there is shown a random-access memory (RAM) structure 70, which may be used to implement the history buffer 24, the hash table 28, the predecessor array 42, and the successor array 44. This type of structure is well known in the art, and it is known to use the simplified notations of FIGS. 2 and 3 to express implementations that can use such structures. Briefly, the conventional memory structure 70 includes a bank 72 of storage elements, addressing logic 74, an address input 76, and a data output 78.

In operation, an address is placed on the address input 76, and the addressing logic 74 selects one of the entries to be presented at the data output 78. This type of random access memory structure can be further controlled by control lines, such as READ lines, WRITE lines, ENABLE lines, and the like, the use of which is well known and will not be discussed further. In one embodiment 16 least significant address bits are used to address locations within the arrays, while more significant address bits select between the arrays.

While the random access memory structure 70 may be used to implement the history buffer 24, the hash table 28, the predecessor array 42, and the successor array 44, in a hardware semiconductor structure, a software implementation of the LZ77 encoding described above may employ higher level storage constructs. For example these may be implemented as data arrays in data memory. As is well known, software array constructs may be stored in semiconductor memory, disk memory, cache memory, special purpose registers, or the like.

Referring now to FIGS. 1, 5 and 13, the segmenter 14 receives (step 260) the output of the LZ77 encoder 12 in a parallel format, and determines whether it represents a match or literal data (step 262), by testing for a match length value of zero. If literal data is detected, a literal token 80 is provided to the Huffman encoder 18 (step 264). A literal token 80 may include a literal token indicator bit sequence 82 followed by literal data 84.

In the case of a match, the segmenter selects either an absolute or difference encoding mode (step 266). This selection is made based on the determination of which mode results in the shortest encoded sequence. The mode chosen affects the point from which the offset is measured, but does not affect the length.

If absolute encoding is selected, the offset is measured between the beginning of the window and the current position in the window, and this offset is provided to the byte counter (step 268). A match token is then provided to the

Huffman encoder (step 270). A match token is made up of a segment number 92 (e.g., 4 bits), and a length 94 (e.g., 6 bits)

Referring also to FIG. 6, the segments 96, 98, 100, 110, 112, 114 used by the segmenter have different lengths, and therefore specifying a position (or segment-based offset) in each of the segments requires a different number of bits. The segments have been arranged such that the shortest segments are most recent and the longest segments are oldest. This takes advantage of the fact that most matches are generally found to occur in the relatively recent past of the history buffer, as illustrated by the exemplary probability distribution curve 116. The first segment 96 will thus have the highest density of matches. Because this is also the shortest segment, these matches will have the shortest segment-based offset indicators. This segmented method of specifying offsets therefore provides added compression over the LZ77 variant.

In the case of difference encoding, the offset is encoded as an offset from the most recent match, even if it is not a match of the same subsequence, and this offset is provided to the byte converter (step 272). A match token is then provided to the Huffman encoder (step 270). The format of this match token is somewhat different from the match token used in absolute encoding.

Referring to FIG. 6, a special segment number is used in difference coding mode. When this segment number is used, offsets are encoded to be interpreted as a difference offset 111 from the last match 109 to the current match 113, rather than from the beginning of a segment boundary. This mode can be particularly useful for compressing in a packet-based environment, where packets which contain different types of data are interleaved onto the communications stream. In such an environment, it is likely that data in a packet will tend to match data in an earlier occurrence of that data type in an earlier packet. These related packets may be separated by a number of intervening packets, however, which could result in a succession of long offsets if the difference coding mode were not used. The offsets in the difference coding mode are not segmented.

Note that the order of the operations may be altered such that both types of offset are first at least partially generated, and these are then evaluated to determine which is shortest. The mode of transmission is then chosen on the basis of this determination.

Referring to FIGS. 1 and 7, once the output of the LZ77 encoder 12 has been processed by the segmenter 14 and converted to tokens, the tokens are encoded by the Huffman encoder 18 to produce Huffman encoded tokens 118 for literal tokens, and 119 for match tokens. These Huffman encoded tokens are provided to the byte converter 19, which adds the appropriate offsets 120 to the Huffman encoded match tokens 119. The offsets are not Huffman encoded, because it has been found that these numbers tend to be quite random and therefore result in little or no compression, while taxing the Huffman encoder.

The Huffman encoded tokens 118, 119 will be of variable length, by definition. The offsets 120 will also be of variable length, due to the segmenting. As the hardware implementation of many computer systems is designed to transfer data in a byte-delineated format, the byte converter breaks the encoded data stream 121 up into bytes for transmission.

Referring to FIGS. 1 and 8, the Huffman encoder 18 is made up of a window or buffer 122, which is responsive to the tokens from the coupling FIFO 16, and Huffman tree storage 124. The tree storage is responsive to tokens from the coupling FIFO and the output of the buffer 122. The Huffman encoder also includes control logic 126 for controlling which symbols are supplied to the Huffman tree storage, and Huffman substitution logic 128, which is responsive to the tokens from the coupling FIFO and the statistical information stored in the Huffman tree storage.

Operation of the Huffman encoder will be discussed in conjunction with FIGS. 1, 8 and 12.

Tokens are received from the segmenter 14, via the coupling FIFO 16, into the buffer 122. Upon reception, each token is substituted for with its current Huffman representation by the Huffman substitution logic 128 (step 250 in FIG. 12). Once this substitution has taken place, the frequency of occurrence of that particular token may be updated in the Huffman tree, which is stored in the Huffman tree storage 124 (step 254). This generally has the effect of updating the Huffman code, so that the next occurrence of the token will usually be encoded with a shorter Huffman code.

As tokens move through the buffer 122 in a FIFO manner, eventually they reach the output of the buffer. At this point, they may be provided to the Huffman tree storage and the frequency for this particular token may be decremented. The manner in which the tree is maintained and updated is known and is discussed further in "Variations On A Theme By Huffman" by Robert G. Gallager, IEEE. Transactions on Information Theory, Vol. IT-24, 6, November 1978, and "Dynamic Huffman Coding" by Donald E. Knuth, Journal of Algorithms 6, 163-180 (1985).

The control logic 126 controls the admission of the tokens from the input and the output of the buffer 122 to the Huffman tree storage. The control logic may admit less than all of the tokens. For example, it may admit only a fraction of the tokens (step 252), such as every four or eight tokens. This sampling may be performed by decrementing a counter each time a token is received, until it reaches zero, at which point a sample is taken. The counter is then reloaded with the sampling ratio.

For sufficient data passing through the buffer, the statistical content of the Huffman tree will be representative of the data encoded, but the standard Huffman updating operations will need to be performed less frequently. This will reduce required processing time significantly, which can result in improved throughput. The sampling ratio used by the control logic may be a user-settable parameter.

One way of viewing the encoding of the tokens is that segment numbers 0 through 3 indicate literal data. The relationship between subsequent segment numbers, the range of offsets for each segment, and the numbers of bits required to offset into the segments for one embodiment are shown in Table I.

Segment	Start	End	Bits
4	0	7	3
5	8	23	4
6	24	87	6
7	88	343	8
8	344	1367	10
9	1368	5463	12
10	5464	32768	15

Table I

A further segment number can be used for difference encoding.

A non-adaptive fixed code may also be substituted for the above adaptive Huffman code, in cases where the adaptive approach would result in degraded performance. This non-adaptive fixed code can shorten the literal data indicator 82 in each literal token 80 to a single bit (see FIG. 6). These modified literal tokens are provided as the output of the encoder along with the match tokens 88 and the variable-length offsets 120.

Referring to FIG. 9, an exemplary decoding system 130 includes a Huffman decoder 132 responsive to a stream of compressed data on the communications channel 22, an unsegmenter 134 responsive to the stream of compressed data and the Huffman decoder, and a string substitution module 136 responsive to the unsegmenter.

In operation, the Huffman decoder 132 decodes Huffman encoded tokens and provides decoded tokens to the unsegmenter 134. The Huffman decoder maintains its frequency information in the same manner as does the Huffman encoder. If the Huffman encoded token represents a literal, the unsegmenter adds the literal data from that token to an output stream 138. If the decoded token represents a match, the unsegmenter reads the segment number so that it can interpret the offset that follows a match token. The unsegmenter then converts the segment number and the segment-based offset into an absolute offset and provides it, along with the length, to the string substitution module 136.
The string substitution module 136 maintains a history buffer, and uses the length and offset to copy matched strings from the history buffer into the output stream.

Note that when the compression system no longer has any data to transmit, its byte converter places an end marker in the string. Upon resumption, the first data received by the decompression system are interpreted as a Huffman encoded token by the decoder and decompression resumes.

A compression system and a decompression system can be provided in the same hardware implementation for bidirectional operation. In software embodiments, these can take the form of different software modules. In hardware embodiments, the encoding and decoding circuits can be separate (full-duplex), or they can share circuitry (half-duplex). Of course, individual compression and decompression systems can each be constructed separately. Furthermore, software and hardware embodiments can communicate with each other. The two encoding stages can also each be used separately.

In one embodiment including features of the invention, a compression system and decompression system share circuitry in an 0.7 micron CMOS gate array, which uses external memory for the history buffer, hash table, and predecessor and successor arrays. The integrated circuit includes control circuitry, registers, hashing circuitry, and a look-ahead buffer. This embodiment has a microprocessor interface and is constructed to interact via DMA with bus-based computer hardware that has an eight or sixteen-bit bus.

In this embodiment, the history buffer may receive the characters from a host system over a bus, by direct memory access (DMA). These characters can be held in a look-ahead buffer provided between the bus and the history buffer, before they are received by the history buffer. This look-ahead buffer can be filled by a DMA controller at the bus' rate, and emptied into the history buffer at a slower rate. The look-ahead buffer therefore allows a computer system employing the compression system to operate somewhat more efficiently, by permitting block transfers of input data to the compression system. An output FIFO may also be provided to serve a similar function during decompression operations.

While there have been shown and described what are at present considered the preferred embodiments of the present invention, it will be obvious to those skilled in the art that various changes and modifications may be made therein without departing from the scope of the invention as defined by the appended claims.

CLAIMS

What is claimed is:

1. Apparatus for compressing a digital sequence comprising:

a first encoder for receiving and compressing the digital sequence according to a first compression procedure to provide a compressed version of the digital sequence,

a buffer responsive to the first encoder to receive and buffer the compressed version of the digital sequence,

a feedback path between the buffer and the first encoder for providing an indication of a fill value for the buffer to the first encoder, wherein the first encoder is responsive to the indication from the buffer to change the first compression procedure,

a statistical data storage element for storing statistical values for symbols in the compressed version of the digital sequence, the statistical data storage element being responsive to the buffer to adjust the statistical values in the data storage element,

control logic for selectively allowing only a subset of the symbols in the compressed version of the digital sequence to adjust the statistical values stored in the data storage element, and

a second encoder responsive to the buffer and to the statistical data storage element, for encoding the symbols in the compressed version of the digital sequence according to a second compression procedure and based on the adjusted statistical values.

2. The apparatus of claim 1 wherein the first encoder comprises:

a history buffer having buffer entries, buffer addressing logic to address one of the buffer entries, and a buffer data output for supplying contents of the one of the buffer entries addressed by the buffer addressing logic, wherein the buffer addressing logic has a predefined number of address lines,

hashing logic responsive to the buffer data output to convert the contents of the one of the buffer entries addressed by the buffer addressing logic to a hash value according to a hashing function, and

a hash table having table entries, table addressing logic responsive to the hashing logic to address one of the table entries with the hashing value, and a table data output for supplying contents of the one of the table entries addressed by the table addressing logic, wherein the table addressing logic has the same predefined number of table address lines as the history buffer has buffer address lines, and wherein the buffer address lines are responsive to the table data output to indicate a location of a match in the history buffer.

3. The apparatus of claim 1 wherein the first encoder comprises:

a history buffer for receiving and buffering the digital sequence,

match detection logic responsive to the history buffer, for detecting matches between a present subsequence in a history buffer and a previous subsequence in a history buffer, and

coding logic responsive to the match detection logic to selectively encode a matched present subsequence as either an absolute offset referenced to the previous subsequence or a relative offset referenced to an earlier match.

4. The apparatus of claim 1 wherein the first encoder comprises:

a history buffer for receiving and buffering the digital sequence, match detection logic responsive to the history buffer, for detecting matches between a present subsequence in a history buffer and a previous subsequence in a history buffer, and

control logic for identifying a portion of the history buffer as a valid portion, the control logic further being responsive to the match detection logic to determine whether the matches detected by the match detection logic are within the valid portion of the history buffer.

5. Apparatus for compressing a digital sequence, comprising:

a first encoder for receiving and compressing the digital sequence according to a compression procedure to provide a compressed version of the digital sequence,

a buffer responsive to the first encoder to receive and buffer the compressed version of the digital sequence, and

a feedback path between the buffer and the first encoder for providing an indication of a fill value for the buffer to the first encoder, wherein the first encoder is responsive to the indication from the buffer to change the compression procedure.

6. The apparatus of claim 5 wherein the first encoder includes storage for a compression parameter and wherein the storage is responsive to the feedback path to adjust the compression parameter in response to the indication from the buffer.

7. The apparatus of claim 6, wherein the first encoder includes a history buffer, and a mechanism for searching for matches within the history buffer, the mechanism being responsive to the storage to limit the amount of search in the history buffer according to the compression parameter.

8. The apparatus of claim 7, wherein the mechanism for searching for matches includes hashing logic, a hashing table, and linked lists depending from the hashing table and wherein the compression parameter limits the depth of search in the linked lists.

9. The apparatus of claim 8 wherein the feedback path is constructed and adapted to provide as the indication a signal indicating that the buffer contains more than a certain number of entries.

10. The apparatus of claim 5 wherein the feedback path is further constructed and adapted to provide as the indication a signal indicating that the buffer contains more than a certain number of entries.

11. The apparatus of claim 5 wherein the feedback path is further constructed and adapted to provide as the indication a signal indicating that the buffer contains fewer than a certain number of entries.

12. A method of compressing a digital sequence, comprising:

encoding the sequence to provide a compressed version of the sequence according to a compression procedure,

buffering the compressed sequence, and

altering the compression procedure used in the step of encoding according to the extent of the buffering in said buffering step.

13. The method of claim 12 wherein the step of encoding searches for matches in the sequence, and wherein the step of altering changes the amount of searching performed.

14. The method of claim 13 wherein the step of encoding involves searching linked lists for potential matches and

wherein the step of altering changes the depth of searching performed in the linked lists.

15. The method of claim 12 wherein the step of altering depends on the amount of the sequence being buffered being more than a first value and the amount of the sequence being buffered being less than a second value.

16. Apparatus for compressing a digital sequence, comprising:

an input for receiving symbols in the sequence,

a statistical data storage element for storing statistical values for symbols in the digital sequence, the data storage element being responsive to the input to adjust the statistical values in the data storage element,

control logic for selectively allowing only a subset of the symbols in the digital sequence to adjust the statistical values stored in the data storage element, and

an encoder responsive to the input and to the statistical data storage element, for encoding the symbols in the sequence based on the adjusted statistical values.

17. The apparatus of claim 16 wherein the control logic is periodic and allows only a predetermined fraction of the symbols in the digital sequence to adjust the frequency values stored in the data storage element.

18. The apparatus of claim 16 further including a buffer responsive to the input and having an output for outputting symbols in the sequence, and wherein the data storage element is further responsive to the output to adjust the frequency values in the data storage element.

19. The apparatus of claim 18 wherein the data storage element increments the frequency value for each symbol upon receiving an instance of that symbol from the input and

decrements the frequency value for each symbol upon receiving an instance of that symbol from the output.

20. The apparatus of claim 16 wherein the apparatus is constructed to implement Huffman encoding.

21. A method of processing a digital sequence, comprising:

receiving symbols from the digital sequence,

sampling a subset of the received symbols,

updating statistical values for the received symbols from the digital sequence based on the subset of the received symbols sampled in the step of sampling, and

substituting, for each of the received symbols, a token derived from the statistical values updated in the step of updating.

22. The method of claim 21 wherein the step of sampling is performed periodically to allow only a predetermined fraction of the symbols in the received digital sequence to update the frequency values.

23. The method of claim 21 wherein the step of sampling a subset includes sampling the subset at two locations in the received sequence, and wherein the step of updating includes increasing the frequency values based on samples received from one of the locations and decreasing the frequency values based on samples received from another of the locations.

24. The method of claim 21 wherein the step of substituting employs a compression method to derive the tokens from the frequency values.

25. The method of claim 24 wherein the step of substituting employs Huffman encoding to derive the tokens from the frequency values.

26. The method of claim 21 wherein the step of substituting employs a decompression method to derive the frequency values.

27. Apparatus for compressing a digital sequence, comprising:

a history buffer for receiving and buffering the digital sequence,

match detection logic responsive to the history buffer, for detecting matches between a present subsequence in a history buffer and a previous subsequence in a history buffer, and

coding logic responsive to the match detection logic to selectively encode a matched present subsequence as either an absolute offset referenced to the previous subsequence or a relative offset referenced to an earlier match.

28. The apparatus of claim 27 wherein the coding logic further includes logic to generate an indicator for indicating whether the matched present subsequence is encoded as an absolute offset or a relative offset.

29. A method of compressing a digital sequence, comprising:

detecting a first match between a first and a second subsequence in the digital sequence,

detecting a second match between a third and a fourth subsequence in the digital sequence, the first and second subsequences being different from the third and fourth subsequences, and

encoding the fourth subsequence using an offset referenced to the first match.

30. The method of claim 29 wherein the step of encoding is performed in response to a further step of determining, the step of determining including determining whether to

encode the fourth subsequence using an offset referenced to the first match instead of an offset from the beginning of a history buffer.

31. The method of claim 29 further including the step of receiving packet-switched data as the digital sequence.

32. A method of decompressing a digital sequence, comprising:

receiving a first match token,

after receiving the first match token, receiving a second match token including an offset,

determining the position of a subsequence in a history buffer based on the offset and the first match token, and

substituting the subsequence for the second match token in the digital sequence.

33. The method of claim 32 further including the step of reading an offset mode indication from the second match token to determine that the step of determining is to be based on the second match token and the offset instead of being based on the offset alone.

34. Apparatus for compressing a digital sequence, comprising:

a history buffer for receiving and buffering the digital sequence,

match detection logic responsive to the history buffer, for detecting matches between a present subsequence in a history buffer and a previous subsequence in a history buffer, and

control logic for identifying a portion of the history buffer as a valid portion, the control logic further being responsive to the match detection logic to determine whether the matches detected by the match detection logic are within the valid portion of the history buffer.

35. The apparatus of claim 34 wherein the control logic is further operations to control the maximum size of the valid portion of the history buffer.

36. A method of compressing a digital sequence, comprising:

maintaining an insert pointer and a check pointer into a history buffer,

inserting characters at the insert pointer,

accessing the history buffer to search for matches in the history buffer, and

determining whether the step of accessing the history buffer is valid based on the insert and check pointers.

37. The method of claim 36 wherein the step of maintaining includes maintaining the check pointer to set the maximum amount of the history buffer in which the step of determining will determine that the step of accessing is valid.

38. The method of claim 37 further including the step of initializing the buffer by setting the insert and check pointers to be equal.

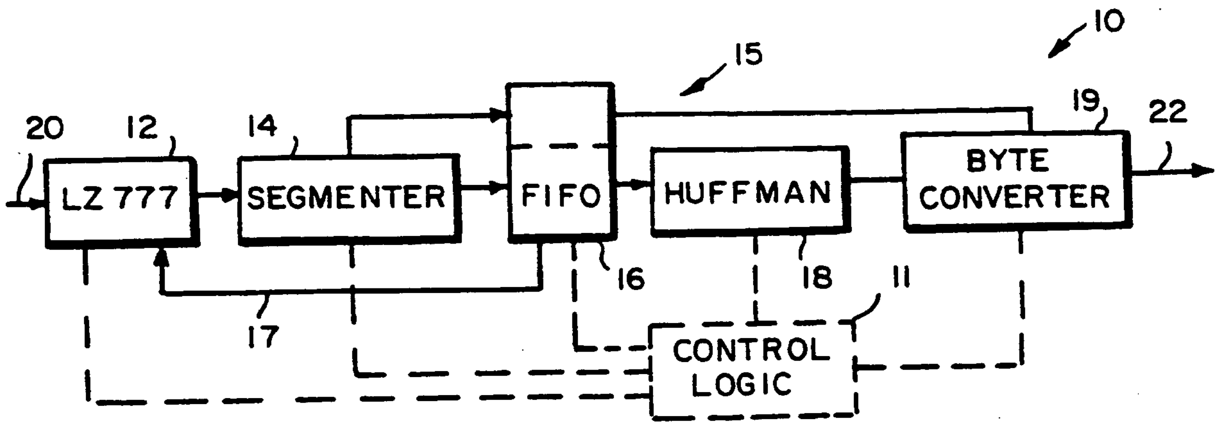

10
15
20
12
14
19
22
LZ 777
SEGMENTER
FIFO
HUFFMAN
BYTE CONVERTER
16
18
11
17
CONTROL LOGIC

FIG. 1

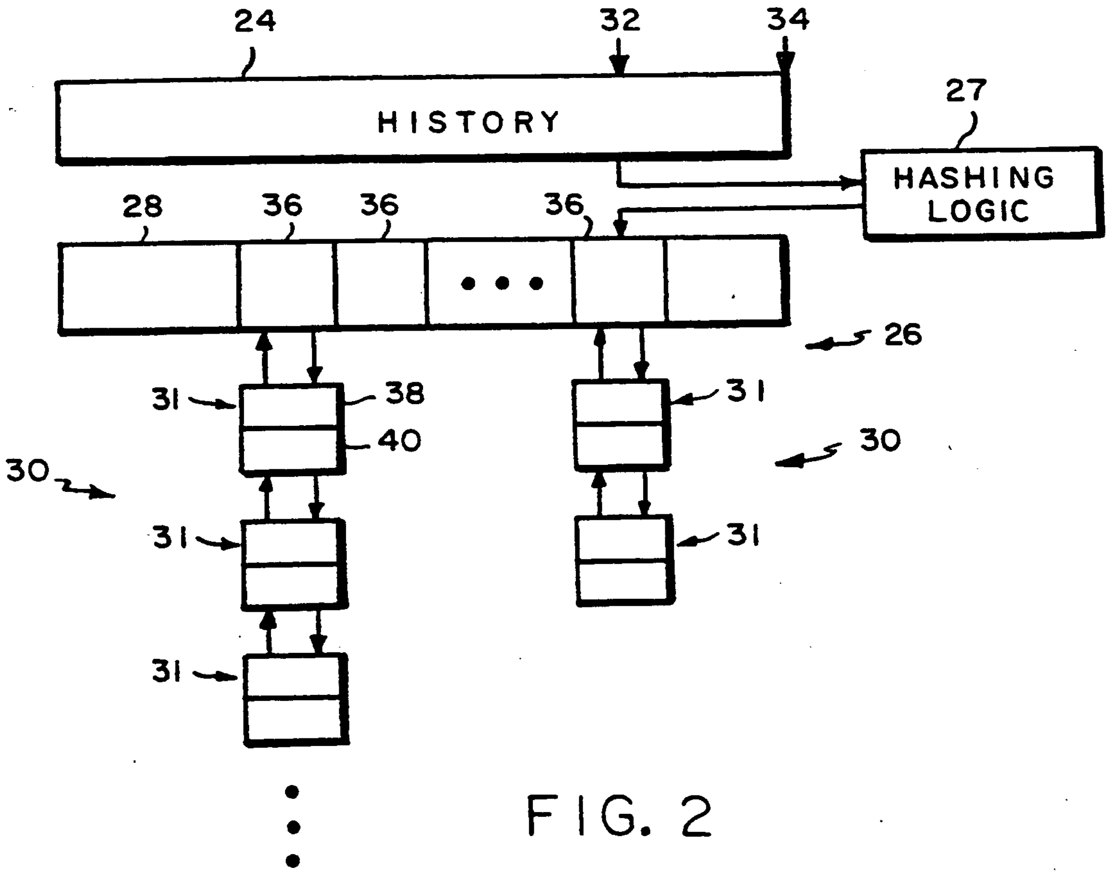

24
32
34
HISTORY
27
HASHING LOGIC
28
36
36
36
26
31
38
40
30
31
30
31
31
31

FIG. 2

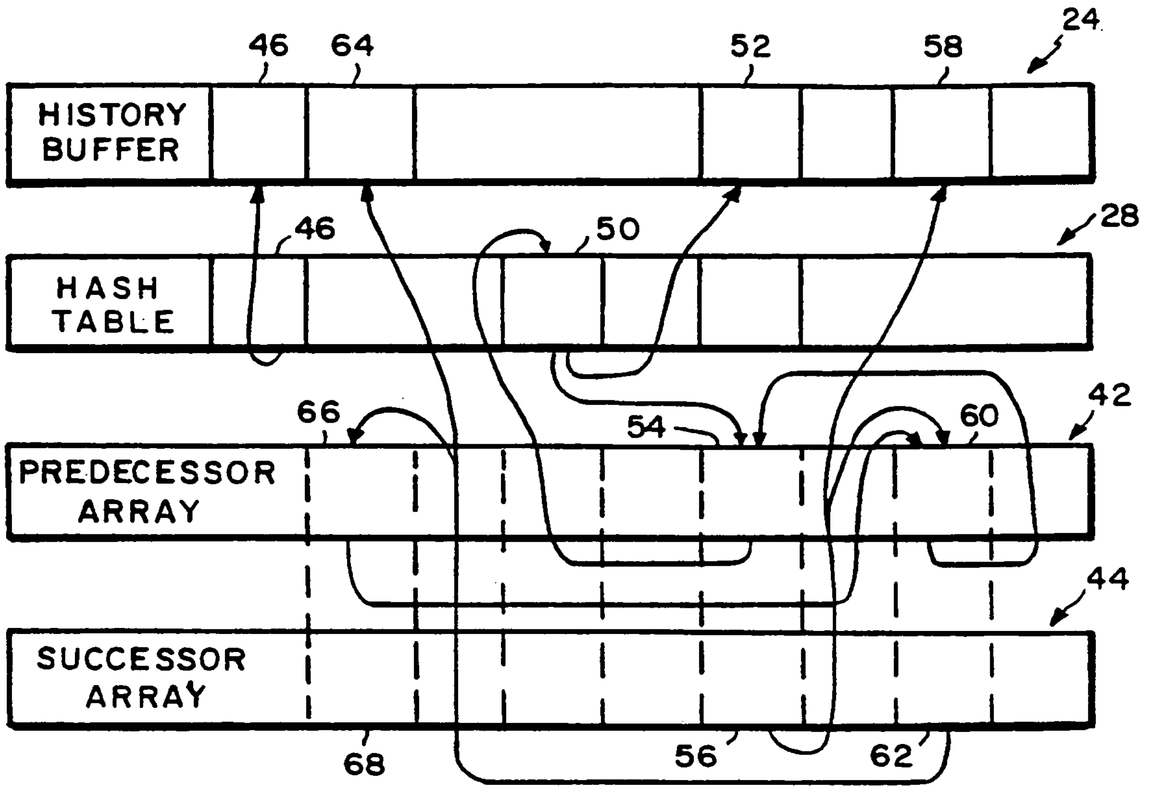

24
46
64
52
58
HISTORY BUFFER
46
50
28
HASH TABLE
66
54
60
42
PREDECESSOR ARRAY
44
SUCCESSOR ARRAY
68
56
62

FIG. 3

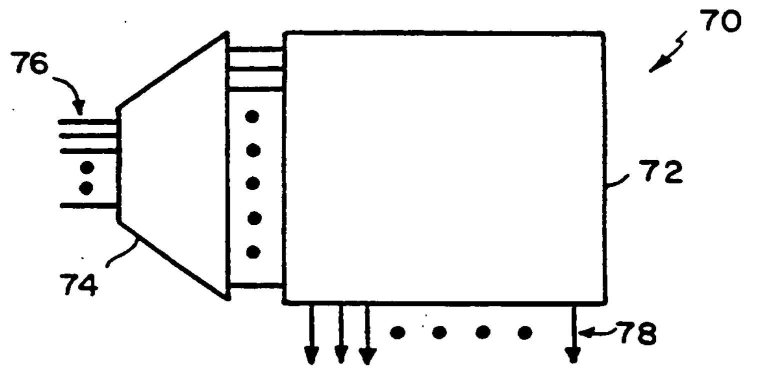

70
76
72
74
78

FIG. 4

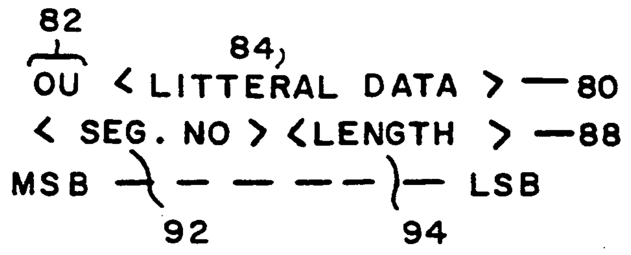


FIG. 5

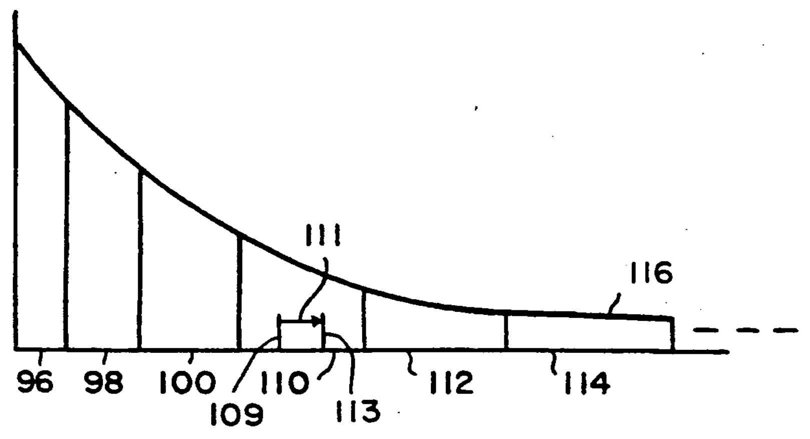


FIG. 6

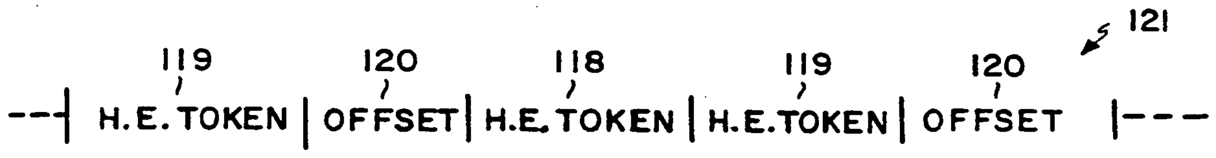


FIG. 7

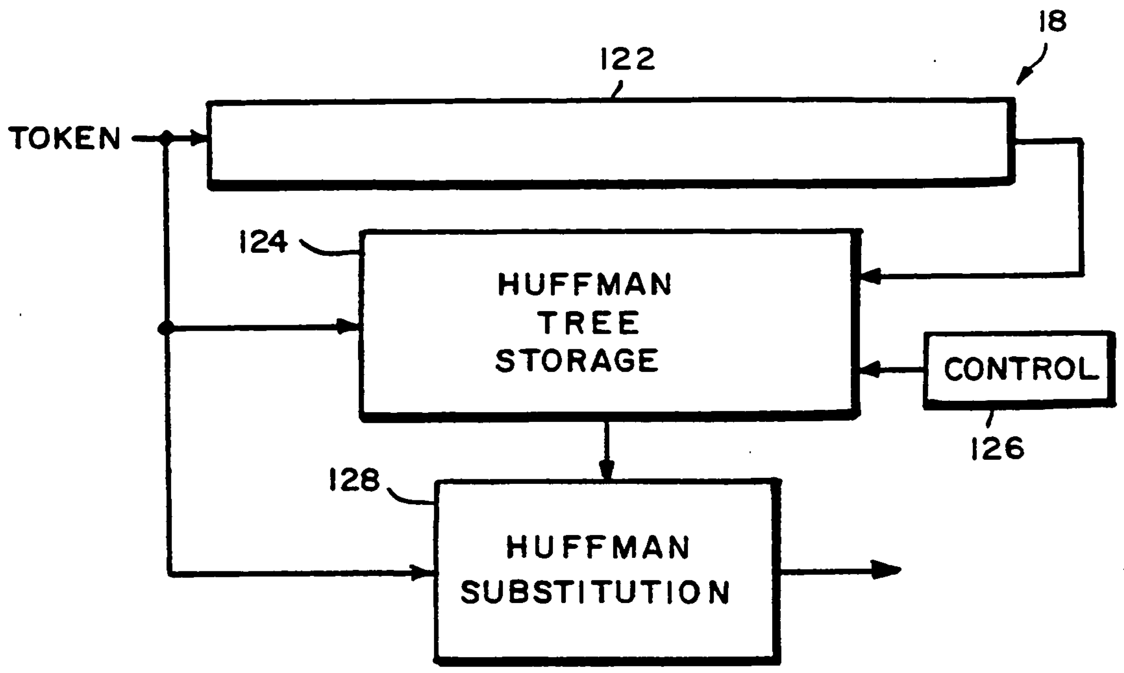

18
122
TOKEN
124
HUFFMAN TREE STORAGE
CONTROL
126
128
HUFFMAN SUBSTITUTION

FIG. 8

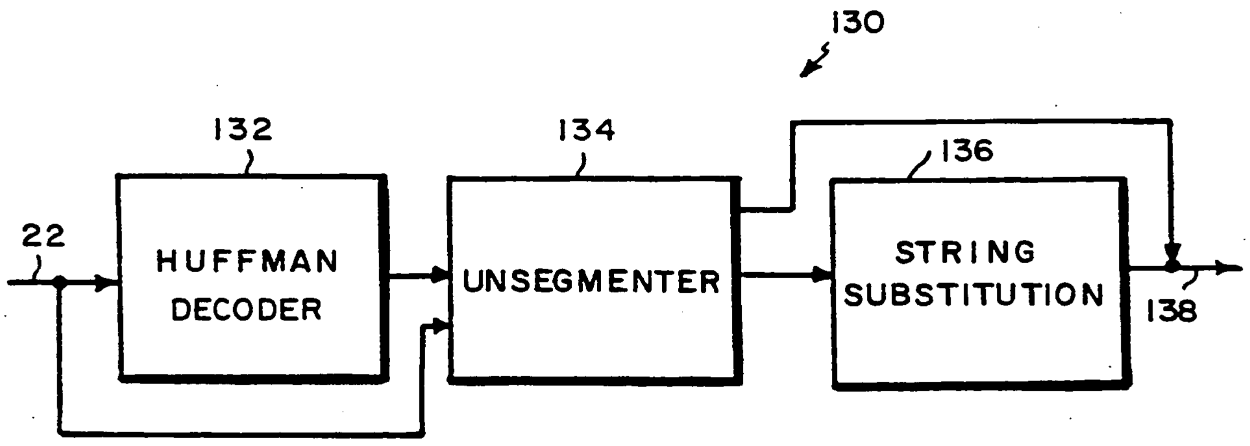

130
132
134
136
22
HUFFMAN DECODER
UNSEGMENTER
STRING SUBSTITUTION
138

FIG. 9

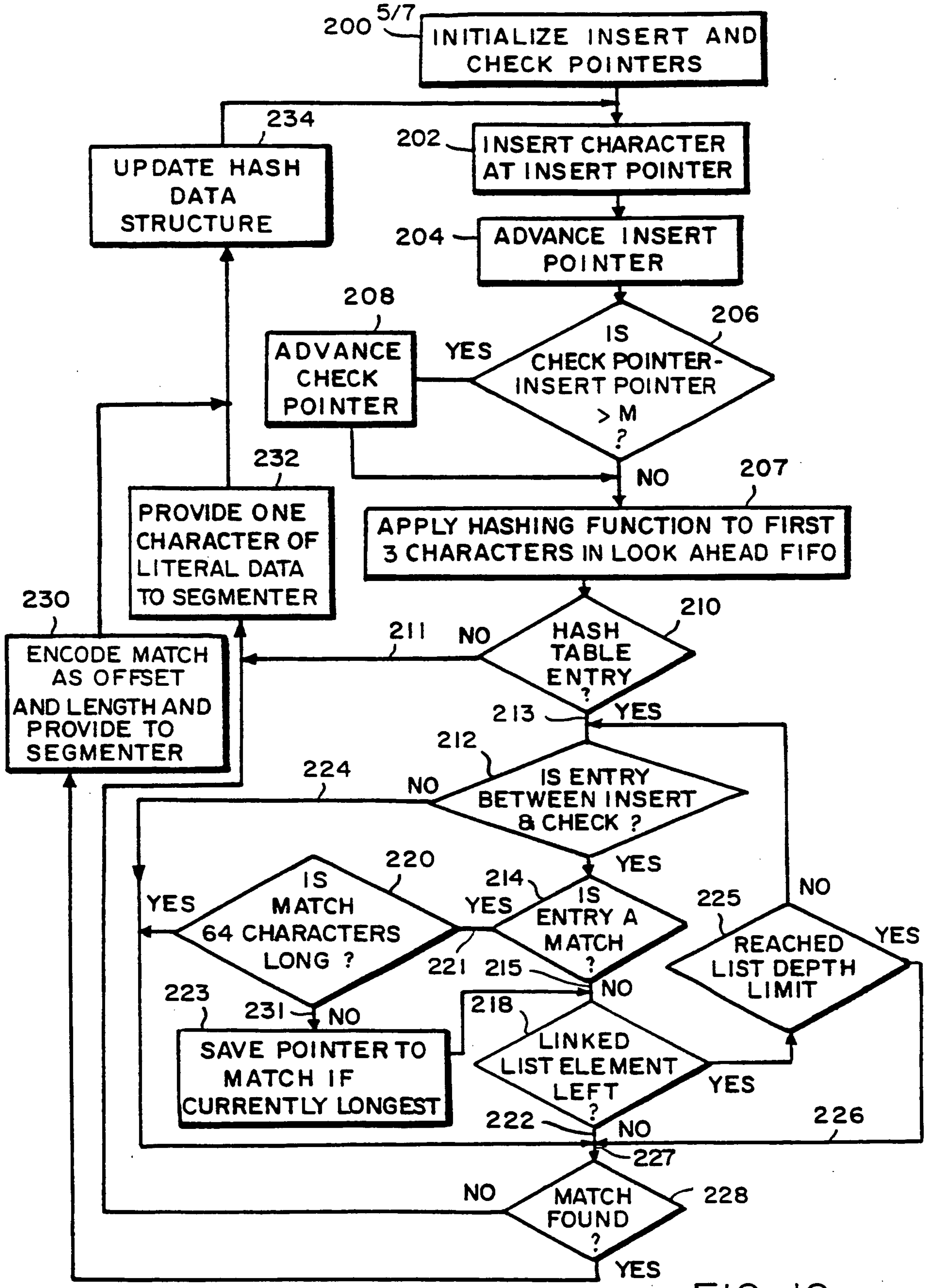

200
INITIALIZE INSERT AND CHECK POINTERS
202
INSERT CHARACTER AT INSERT POINTER
204
ADVANCE INSERT POINTER
206
IS CHECK POINTER-INSERT POINTER > M ?
YES
208
ADVANCE CHECK POINTER
NO
207
APPLY HASHING FUNCTION TO FIRST 3 CHARACTERS IN LOOK AHEAD FIFO
210
HASH TABLE ENTRY ?
211
NO
213
YES
212
IS ENTRY BETWEEN INSERT & CHECK ?
NO
224
YES
214
IS ENTRY A MATCH ?
YES
221
215
NO
220
IS MATCH 64 CHARACTERS LONG ?
YES
231
NO
223
SAVE POINTER TO MATCH IF CURRENTLY LONGEST
218
LINKED LIST ELEMENT LEFT ?
YES
222
NO
225
REACHED LIST DEPTH LIMIT
NO
YES
226
227
228
MATCH FOUND ?
NO
YES
230
ENCODE MATCH AS OFFSET AND LENGTH AND PROVIDE TO SEGMENTER
232
PROVIDE ONE CHARACTER OF LITERAL DATA TO SEGMENTER
234
UPDATE HASH DATA STRUCTURE

FIG. 10

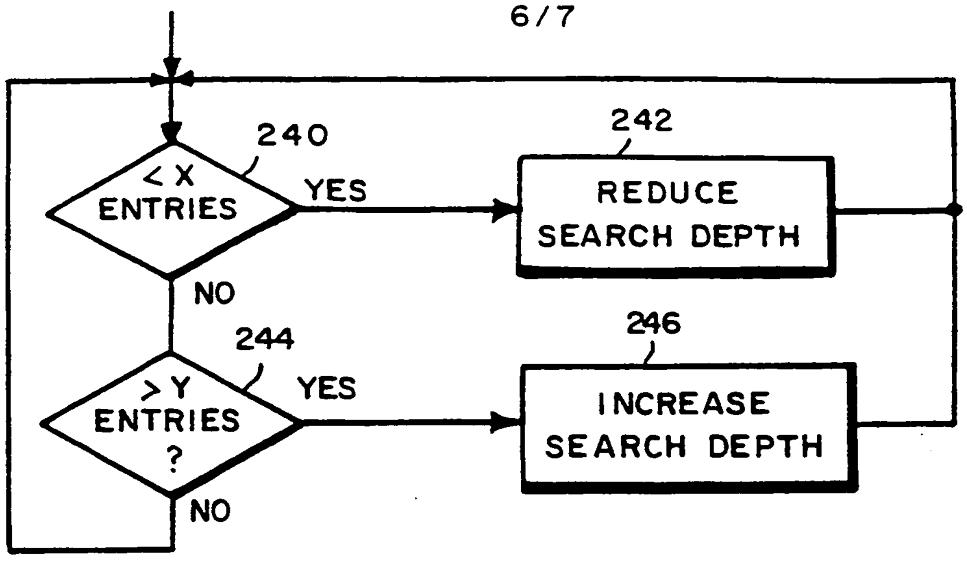

240
< X
ENTRIES
YES
242
REDUCE
SEARCH DEPTH
NO
244
> Y
ENTRIES
?
YES
246
INCREASE
SEARCH DEPTH
NO

FIG. 11

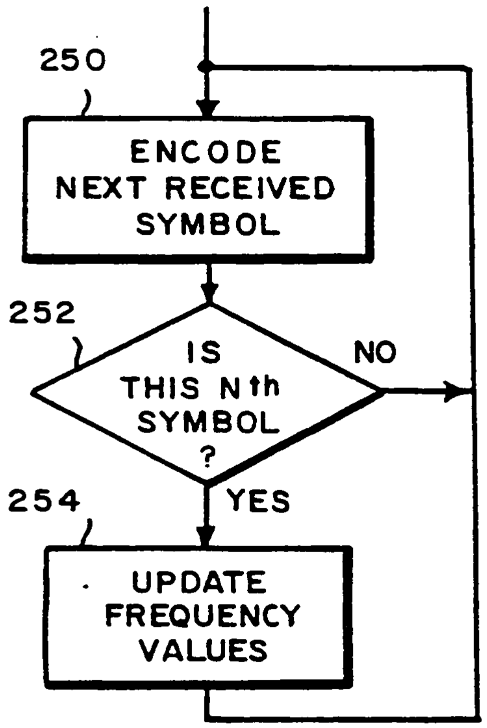

250
ENCODE
NEXT RECEIVED
SYMBOL
252
IS
THIS Nth
SYMBOL
?
NO
254
YES
UPDATE
FREQUENCY
VALUES

FIG. 12

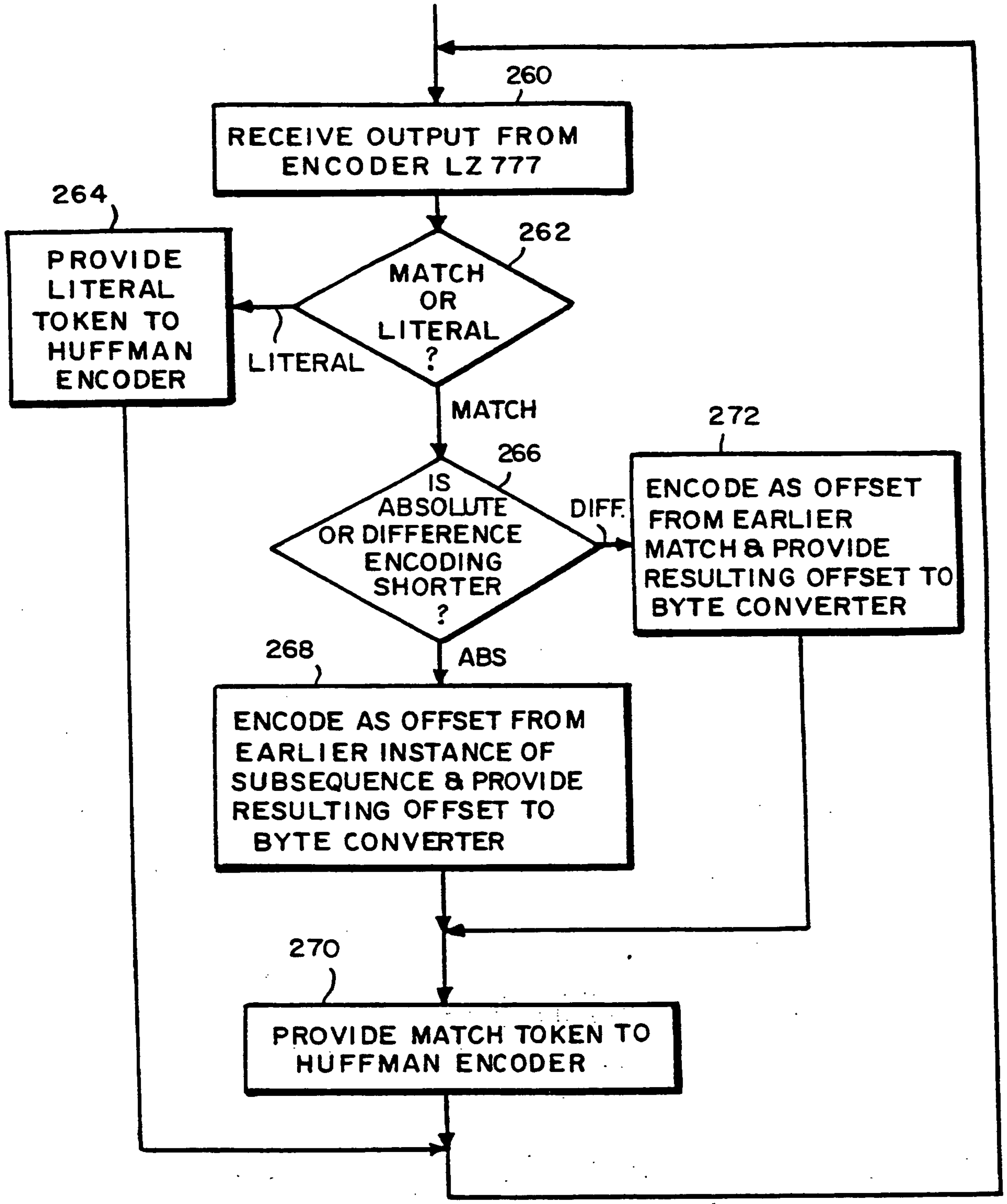

260
RECEIVE OUTPUT FROM ENCODER LZ 777
262
MATCH OR LITERAL ?
LITERAL
264
PROVIDE LITERAL TOKEN TO HUFFMAN ENCODER
MATCH
266
IS ABSOLUTE OR DIFFERENCE ENCODING SHORTER ?
DIFF.
272
ENCODE AS OFFSET FROM EARLIER MATCH & PROVIDE RESULTING OFFSET TO BYTE CONVERTER
ABS
268
ENCODE AS OFFSET FROM EARLIER INSTANCE OF SUBSEQUENCE & PROVIDE RESULTING OFFSET TO BYTE CONVERTER
270
PROVIDE MATCH TOKEN TO HUFFMAN ENCODER

FIG. 13

INTERNA[illegible]ONAL SEARCH REPORT

In[illegible]ional Application No
PCT/US 94/14823

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H03M7/30 H03M7/42

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H03M H04N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP,A,0 309 280 (BRITISH TELECOMMUNICATIONS) 29 March 1989 see column 1, line 1 - column 3, line 25; figure 2 ---	1,5,6, 10-12,15
Y X	BELL, CLEARY AND WITTEN 'text compression' , PRENTICE HALL , ENGLEWOOD CLIFFS NJ (US) see page 122, line 35 - line 36 ---	1,5,6, 10-12,15 16, 20-22, 24,25
X	US,A,5 159 336 (RABIN ET AL) 27 October 1992 see column 2, line 26 - column 9, line 14; figures 1-5 --- -/--	5-7,9-15

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search 6 April 1995	Date of mailing of the international search report 03. 05. 95
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Feuer, F

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 94/14823

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US,A,5 140 321 (JUNG) 18 August 1992 see the whole document ---	2,27,32, 34
A	WO,A,87 05178 (DEUTSCHE THOMSON-BRANDT GMBH) 27 August 1987 see page 6 -----	16,17,20

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 94/14823

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0309280	29-03-89	AU-B-	637284	20-05-93
		AU-A-	1591992	25-06-92
		AU-A-	2489288	18-04-89
		CA-A-	1330122	07-06-94
		DE-A-	3882469	26-08-93
		DE-T-	3882469	11-11-93
		FI-B-	91474	15-03-94
		WO-A-	8903153	06-04-89
		JP-T-	2501436	17-05-90
		NO-B-	175558	18-07-94
		US-E-	RE34824	10-01-95
		US-A-	4985766	15-01-91
US-A-5159336	27-10-92	NONE		
US-A-5140321	18-08-92	US-A-	5281967	25-01-94
WO-A-8705178	27-08-87	DE-A-	3605032	20-08-87
		EP-A,B	0258341	09-03-88
		JP-A-	62258560	11-11-87
		US-A-	4839724	13-06-89